Workflow Service Extensions for UNICORE 6 – Utilising a Standard WS-BPEL Engine for Grid Service Orchestration

S. Gudenkauf[1], W. Hasselbring[1], A. Höing[2], G. Scherp[1], and O. Kao[2]

[1] OFFIS Institute for Information Technology, R&D-Division Business Information Management, Escherweg 2, 26121 Oldenburg, Germany
{stefan.gudenkauf,guido.scherp}@offis.de,
hasselbring@informatik.uni-oldenburg.de

[2] Technische Universität Berlin, Faculty IV - Electrical Engineering and Computer Science, Dept. of Telecommunication Systems, Complex and Distributed IT Systems, Einsteinufer 17, 10587 Berlin, Germany
{andre.hoeing,odej.kao}@tu-berlin.de

Abstract. The BIS-Grid project[1], a BMBF-funded project in the context of the German D-Grid initiative, focusses on realising Enterprise Application Integration using Grid technologies to proof that Grid technologies are feasible for information systems integration. Small and medium enterprises shall be enabled to integrate heterogeneous business information systems and to use external Grid resources and services with affordable effort.

In this paper, we describe service extensions to UNICORE 6 to use an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate stateful, WSRF-based Web Services, also called Grid Services. Thereby, we focus on how to combine the arbitrary workflow engine with UNICORE 6, and on how to access workflows and workflow instances. The workflows itself are also provided as Grid Services, realised by a Workflow Management Service that deploys Workflow Services within UNICORE 6, each wrapping a WS-BPEL workflow that is deployed in the arbitrary workflow engine.

1 Introduction

In order to map business processes to the technical system level the integration of heterogeneous information systems - referred to as Enterprise Application Integration (EAI) - is crucial. Thereby, integration is often achieved by service orchestration in service-oriented architectures (SOA). A means commonly used to create SOA are Web Services since they enable service orchestration and hide the underlying technical infrastructure. Modern Grid middlewares such as

[1] This work is supported by the German Federal Ministry of Education and Research (BMBF) under grant No. 01IG07005 as part of the D-Grid initiative.

E. César et al. (Eds.): Euro-Par 2008 Workshops, LNCS 5415, pp. 103–112, 2009.

UNICORE 6[2] and Globus Toolkit 4[3] are based on the Web Service Resource Framework (WSRF) [1], a standard that extends classical, stateless Web Services to be stateful. Such WSRF-based Web Services, also called Grid Services, provide a basis to build SOAs using Grid technologies.

In BIS-Grid we focus on realising EAI using Grid technologies. One major objective is to proof that Grid technologies are feasible for information systems integration. Small and medium enterprises (SMEs) shall be enabled to integrate heterogeneous business information systems and to use external Grid resources and services with affordable effort. To do so, we develop a workflow engine, the *BIS-Grid workflow engine*, that is capable to integrate Grid Services. This engine is based upon service extensions to the UNICORE 6 Grid middleware, using an arbitrary WS-BPEL workflow engine and standard WS-BPEL to orchestrate Grid Services. Also, it propagates service orchestrations as Grid Services. The main reason that led us to the decision to use UNICORE 6 is that UNICORE 6 is a pioneer in adopting Grid standards, since the support of standards is essential for us, especially regarding security. The WS-BPEL workflow engine to be used is ActiveBPEL[4] since it exhaustively supports the WS-BPEL standard, and is well accepted in the business domain as well as in the Grid domain. We refrain from extending well-adopted standards and technologies as far as possible to increase sustainability. Instead, we use service extensions to UNICORE 6 to conceal the WS-BPEL engine by wrapping the message exchange between the engine and Grid Services.

This paper presents a snapshot on our ongoing work and is organised as follows. Related work is presented in Section 2. In Section 3, we present the architecture of the BIS-Grid solution in detail, highlighting the service extensions to UNICORE 6. Finally, in Section 4, we conclude the paper and briefly present our future work.

2 Related Work

Orchestrating stateful Grid Services is in the focus of many German and international projects, for example, the German D-Grid projects Text-Grid[5] and InGrid[6], and the European projects A-WARE[7], Chemomentum[8], and EGEE[9]. Orchestrating Grid Services is also in the focus of several papers. Leymann [8] describes the appropriateness of using BPEL4WS as a basis for Grid Service orchestration since it already fulfils many requirements of the WSRF standard. He concludes that a Grid-specific extension of BPEL4WS is more appropriate

[2] http://www.unicore.eu
[3] http://www.globus.org/toolkit/
[4] http://www.activevos.com/community-open-source.php
[5] http://www.textgrid.de/
[6] http://www.ingrid-info.de/
[7] http://www.a-ware-project.eu/
[8] http://www.chemomentum.org
[9] http://www.eu-egee.org/

than creating new Grid-specific standards. The appropriateness of BPEL4WS for Grid Service orchestration is also examined and confirmed in [5], [10], [2], and [6]. In [4], Dörnemann et al. discuss composing Grid Services by using BPEL4WS. They present a solution that is based on extending the BPEL4WS specification. Emmerich et al. [5] describe the evaluation of reliability, performance, and scalability issues of the open source workflow engine ActiveBPEL on executing a complex scientific Grid workflow. As a preparatory work to this paper, we describe the requirements that apply to a Grid-enabled workflow system in [6].

UNICORE itself provides a workflow system extension for an existing UNICORE 6 installation, originating from the Chemomentum project. This system consists of two UNICORE/X service containers: a workflow engine, processing workflows on a logical level, and a service orchestration layers that concerns the invocation of Grid Services. The workflow engine utilises pluggable domain-specific language (DSL) modules and a generic workflow language internally. It also includes a resource brokering component. Both the UNICORE 6 workflow system and the BIS-Grid workflow engine have in common that they are realised as service extensions to the UNICORE/X service container. However, the UNICORE 6 workflow system does not feature the integration nor the exchangeability of an arbitrary WS-BPEL workflow engine and is therefore less sustainable.

The works presented in the preceding paragraphs mainly focus on scientific workflows instead of business workflows relevant to BIS-Grid. Concerning the use of BPEL (BPEL4WS or WS-BPEL), it is primarily relied on extending or adapting the language, thus creating BPEL dialects. The A-WARE project is one project that addresses business workflows in Grid environments. In [3], the project presents the orchestration of UNICORE 6 services with the help of a standard BPEL engine, relying on a Service Bus that supports adapters to submit jobs to UNICORE. However, workflows are not provided as Grid Services.

When regarding business workflows, another important aspect is choreography. Choreography regards the interaction and cooperation of multiple workflows, potentially traversing organisational borders. Choreography is relevant to Grid environments since Grid-based Virtual Organisations (VOs) may require workflow cooperation. The TrustCoM[10] project, for example, provides a trust management framework for the definition and enactment of collaborative business processes within VOs. BIS-Grid instead focusses on orchestration. It regards itself as a a door opener for SMEs, allowing them to integrate their information systems with technologies that are Grid-compatible, also allowing them to integrate external Web and Grid Services. Nevertheless, this does not prohibit to consider choreography in future.

3 UNICORE 6 Service Extensions

Our UNICORE 6 service extensions mainly consists of two WSRF service types, *Workflow Management Service* and *Workflow Service*, and an arbitrary standard

[10] http://www.eu-trustcom.com/

WS-BPEL workflow engine. In our case this is the open source workflow engine ActiveBPEL. Together, the service extensions and the arbitrary WS-BPEL engine represent the *BIS-Grid workflow engine*. The service extensions are realised as Grid Services within UNICORE 6's service container, the UNICORE/X component. For each workflow deployed with the Workflow Management Service one Workflow Service will be created using a hot deployment mechanism without restarting UNICORE/X. These services manage and access ActiveBPEL. As a standard WS-BPEL workflow engine, it typically orchestrates stateless Web Services and supports only basic security mechanisms, e.g. username-based and password-based authentication. Therefore, advanced security concepts must be provided by the service extensions in the UNICORE/X service container. In [7] we illustrate some considerations on security within the BIS-Grid solution.

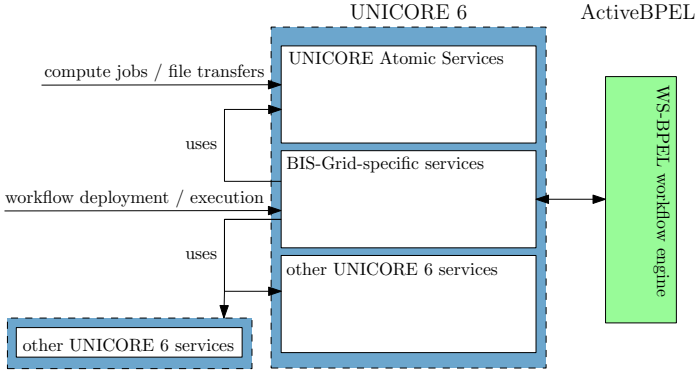
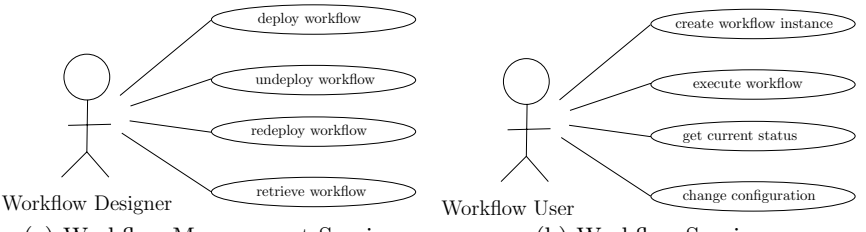


Fig. 1. Overview on the Architecture of the BIS-Grid solution

Figure 1 presents an overview on the architecture of the BIS-Grid workflow engine. Within UNICORE/X, the BIS-Grid service extensions are placed beside so-called *UNICORE Atomic Services* which provide basic functionalities to support Grid computing, and beside other Grid Services that, e.g. may provide access to information systems. One important design decision was to neither extend the WS-BPEL standard nor to modify ActiveBPEL for Grid Service orchestration, although the WS-BPEL 2.0 specification provides an extensibility mechanism that allows to integrate additional functionality without declining the standard. However, the use of proprietary extensions would conclude in a solution that may not be interoperable with future versions of the standard as well as with the engine.

Leaving the WS-BPEL standard and the engine untouched ensures sustainability and flexibility, and allows to exchange the WS-BPEL engine by any other WS-BPEL engine. Figure 1 shows that the ActiveBPEL engine is located behind UNICORE 6. Hence, it can be deployed separately on backend nodes to support load balancing. In [7] we also present our considerations on load balancing the BIS-Grid solution.

Besides all this advantages also some problems arise when using such a decoupled architecture without WS-BPEL extensions. The BPEL code, that is necessary to call a Grid Service is more complex as if we would introduce new grid-specific activities. Even, if a WSRF-resource of a Grid Service is only used for a single call, we have to explicitly create and destroy it using BPEL invoke activities. We tackle this problems by hiding the complexity from the user and the as far as possible from the workflow designer. We plan to extend an existing BPEL editor by introducing new "Grid Activities" that automatically generates the BPEL code. Furthermore, we need some additional BPEL code to solve a mapping problem, that is described in more detail in Section 3.3. The mapping problem will be hidden completely in the Workflow Management Service, that will do some processing steps before the deployment of the BPEL code to the WS-BPEL workflow.



(a) Workflow Management Service (b) Workflow Service

Fig. 2. Use Cases

3.1 Workflow Management

Important workflow management functionalities are workflow deployment, redeployment, undeployment, and retrieval (see Figure 2(a)). These functionalities are realised as a Grid Service, the *Workflow Management Service*, whereas an instance of the service manages exactly one workflow. Workflow Management Service instances are created by a factory that is realised as a standard Web Service within the UNICORE/X service container. This factory offers the following functions:

- `create`: The method creates a new (empty) service instance and returns the corresponding endpoint reference.
- `search`: The method returns a set of endpoint references pointing to service instances that represent workflows accessible to the user. Search patterns can be used to limit the result list.

A Workflow Management Service instance provides the following functions:

- `deploy`: The method deploys a workflow by requiring a *BIS-Grid Workflow Deployment Package*. This package contains a WS-BPEL workflow description and additional resources, e.g. deployment descriptors. Since each Workflow Management Service instance manages only one workflow this method

is blocked after deployment until the `undeploy` method has been called. To modify a deployed workflow the method `redeploy` must be used.
- `undeploy`: This method undeploys a workflow previously deployed with the same Workflow Management Service instance, and destroys the corresponding Workflow Service.
- `redeploy`: This method redeploys a workflow previously deployed with the same Workflow Management Service instance. The effect is regarded the same as if the methods `undeploy` and `deploy` are called consecutively.
- `retrieve`: This method returns the corresponding deployment package of a workflow previously deployed with the same Workflow Management Service instance.

For workflow deployment and undeployment, the Workflow Management Service has to communicate with the WS-BPEL engine and with the UNICORE/X service container to create or remove Workflow Services (see Section 3.2). Therefore, the deployment and undeployment processes can be subdivided into several steps. If one step fails the complete deployment or undeployment process must fail, too, and the preceding steps must be rolled back (if required and possible). The deployment process is as follows:

1. The BIS-Grid Workflow Deployment Package is stored to a previously specified local file space and is unpacked.
2. The deployment package, containing the workflow description and a corresponding deployment descriptor, is checked for correctness and completeness.
3. The WS-BPEL workflow description is modified to solve a UNICORE 6/WS-BPEL workflow engine mapping problem. Details about this problem and the solution are described in Section 3.3.
4. The workflow is deployed to the WS-BPEL workflow engine. This is done by transferring the WS-BPEL workflow description and the corresponding deployment descriptor to the WS-BPEL engine by an appropriate adapter, in our case an ActiveBPEL adapter.
5. The corresponding Workflow Service (see Section 3.2) is created and registered to the UNICORE/X service container.

Subsequently, undeployment is executed as follows:

1. The factory service of the corresponding Workflow Service (see Section 3.2) is deregistered and removed from the UNICORE/X service container to prevent the creation of new Workflow Service instances.
2. The undeployment process waits for the termination of active workflow instances. The initiator of undeployment may decide whether these instances shall terminate normally (expiration date is infinite), instantly (expiration date is *0*), or at a specific date (expiration date is specified either explicitly or by a grace time). Except for normal termination, Workflow Service instance termination is enforced by the undeployment process at the given expiration date. By default, the normal termination strategy is used.
3. The actual Workflow Service (see Section 3.2) is deregistered and removed from the UNICORE/X service container.

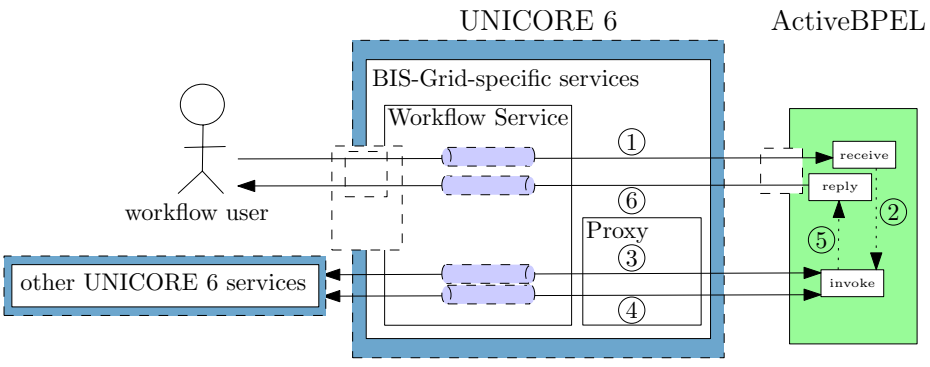


Fig. 3. Workflow Service Architecture and Example Usage

4. The WS-BPEL workflow is undeployed from the WS-BPEL workflow engine by using an appropriate adapter (ActiveBPEL). It has to be ensured that all data concerning the WS-BPEL workflow to be undeployed is removed.
5. The BIS-Grid Workflow Deployment Package and all related data are removed from the respective local file space.

Beside the functionalities described in this section, WS-BPEL engine management, and high-level monitoring and auditing of Workflow Service instances can also be seen as a part of workflow management. Appropriate services are already envisioned in our architecture but are not part of this work.

3.2 Workflow Service

A Workflow Service represents the execution of a specific deployed workflow. It is a WSRF service which instances are created through a corresponding factory service. Each instance is mapped directly to one workflow instance in the ActiveBPEL workflow engine. The Workflow Service must fulfil the use case shown in Figure 2(b): workflow instance creation, workflow execution, status information providing, and online configuration modification.

Figure 3 shows a more detailed view on a Workflow Service (note that the factory service is omitted). We assume a simple workflow that calls an external service and sends the response back to the user. Only relevant WS-BPEL activities are shown in the ActiveBPEL engine. Before the execution starts, the user has to create a new Workflow Service instance via the corresponding factory service.

The WSDL interface of this Workflow Service is a combination of two interfaces: The first one is the Workflow Service interface that offers BIS-Grid-specific operations, e.g. requesting information on workflow progress, changing/adding security tokens, or modifying the security policy. The second interface is the original workflow interface provided by the WS-BPEL engine integrated in the combined WSDL so that all service calls can be done on the Workflow Service.

Internally, the UNICORE/X service container contains an XFire SOAP engine[11] that processes incoming messages through an XFire Handler Pipeline. The last handler of the incoming pipeline is the so-called *Invoker* that manages the Java method invocation on the actual UNICORE/X service. The Invoker of a Workflow Service instance differentiates between BIS-Grid-specific service calls and service calls specific to workflow execution. Regarding the latter, the Invoker forwards the message to a general method. Grid-specific information such as security credentials or accounting and billing information is removed in a second configurable handler pipeline, thus converting the message to a standard message used in Web Service calls (see Figure 3 (1)). This will be realised using a separate handler pipeline. The BPEL workflow engine works up the message (2).

Vice versa, messages sent from the WS-BPEL workflow to invoke external services (3) are caught by a *proxy* also located in UNICORE/X. The proxy reads an identifier consisting of the name of the workflow and the id of the corresponding Workflow Service instance (cp. Section 3.3) into the message header, and forwards the message to the correct Workflow Service instance for further processing. By using another handler pipeline that depends on the current configuration of the Workflow Service instance, Grid-specific XML fragments - e.g. SAML assertions [9] - are added to the message and then forwarded through a certificate-secured SSL channel. In case of a synchronous service call the answer is subsequently processed in reverse, removing and processing Grid-specific XML fragments and forwarding it to the WS-BPEL engine through the connection held by the proxy (4). Regarding asynchronous service calls the connection will be closed after the message is sent.

The response message from the external service is also worked up by the BPEL engine (5) and the answer is send back to the user (6).

ActiveBPEL runs in a separate environment. It is not possible to access the engine without using the BIS-Grid-specific services of a UNICORE 6 installation. This is crucial for Grid workflow execution security since workflow access is controlled by UNICORE 6 security mechanisms using Grid credentials. All message transfers are performed using certificate-secured SSL channels.

3.3 UNICORE 6/WS-BPEL Engine Mapping

Using an arbitrary WS-BPEL engine, we have to deal with two different instances of the same workflow: a regular workflow instance in the actual WS-BPEL engine, and the WSRF Workflow Service instance in the UNICORE/X service container. It is the task of the UNICORE/X service extensions to check incoming and outgoing messages and to prepare them for Grid utilisation. To do so, it is necessary to map messages from the WS-BPEL engine to the correct WSRF instances and vice versa. In UNICORE 6, a Workflow Service instance is identified by a resource id which is also contained in its endpoint reference. Unfortunately, there is no identification information in the SOAP messages that are sent between UNICORE/X and the WS-BPEL engine which enables to

[11] http://xfire.codehaus.org/

conclude the original WS-BPEL instance. Such information is necessary for both synchronous and asynchronous external service invocations, e.g. since appropriate security credentials must be assigned to outgoing messages.

This problem is solved by modifying the WS-BPEL workflow description so that the resource identifier used by UNICORE/X is known to the WS-BPEL workflow and used in external message communication (cp. Section 3.1). Therefore, we stipulate that a WS-BPEL workflow expects the resource id of a corresponding Workflow Service instance to be attached to incoming messages that create workflow instances. Second, we stipulate an `assign` operation before each external service invocation within the WS-BPEL workflow description. This `assign` operation inserts the resource id into the message. All in all, the following requirements (RQ) must be met: For all *start*-messages[12] the corresponding XML schema must be extended by an extra variable for the resource id identifying the UNICORE/X Workflow Service instance (RQ1). There must be a dedicated process variable within the workflow to store the resource id during the whole process execution (RQ2). There must be an `assign` activity after each instance-creating activity which copies the resource id from the message into the process variable (RQ3). All outgoing message schemas, i.e. `reply` or `invoke`, must be extended by an extra variable to carry the resource id (RQ4). There must be an `assign` activity that copies the resource id from the dedicated process variable into the corresponding message variable before each activity that causes an outgoing message (RQ5). We developed WS-BPEL patterns that meet these requirements. They will be part of a WS-BPEL pattern catalogue that also contains patterns for Grid utilisation with standard WS-BPEL, for example.

3.4 Implementation Status

A first prototype of the BIS-Grid workflow engine is expected to be released in August 2008. The prototype and the WS-BPEL pattern catalogue mentioned in Section 3.3 will be made available on the BIS-Grid web site (www.bisgrid.de).

4 Conclusion and Future Work

We presented an overview on the architecture of the BIS-Grid workflow engine. It mainly consists of two Grid Services working together with the WS-BPEL engine ActiveBPEL. Instances of a Workflow Management Service hot-deploy Workflow Services that encapsulate workflows in the WS-BPEL engine, and propagate them as WSRF-compliant Grid Services. Within this architecture, neither the WS-BPEL engine nor the WS-BPEL 2.0 standard have to be adapted. The WS-BPEL engine is therefore exchangeable through any WS-BPEL-compliant engine. Also, the architecture offers further possibilities to easily adopt future features, e.g. by modifying handler pipelines.

[12] I.e. messages which are addressed to `receive` or `pick` activities with the `createInstance` attribute set to *yes*.

This paper presents a snapshot on our ongoing work. We recently started with the implementation, focussing on the basic functionalities, e.g. piping messages through the Workflow Service and on security issues. We will subsequently address human interaction in worflows, and consider an adequate workflow design tool in our future work.

References

1. Tim Banks. Web Services Resource Framework (WSRF) - Primer v1.2 (May 2006), http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf
2. Kuo-Ming, C., Younas, M., Griffiths, N., Awan, I., Anane, R., Tsai, C.-F.: Analysis of Grid Service Composition with BPEL4WS. In: Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA 2004), vol. 01, p. 284. IEEE Computer Society, Los Alamitos (2004)
3. Clementi, L., Cacciari, C., Melato, M., Menday, R., Hagemeier, B.: A Business-Oriented Grid Workflow Management System. In: Bougé, L., Forsell, M., Träff, J.L., Streit, A., Ziegler, W., Alexander, M., Childs, S. (eds.) Euro-Par Workshops 2007. LNCS, vol. 4854, pp. 131–140. Springer, Heidelberg (2008)
4. Dörnemann, T., Friese, T., Herdt, S., Juhnke, E., Freisleben, B.: Grid Workflow Modelling Using Grid-Specific BPEL Extensions (2007)
5. Emmerich, W., Butchard, B., Chen, L., Price, S.L., Wassermann, B.: Grid Service Orchestration Using the Business Process Execution Language (BPEL). Journal of Grid Computing, 283–304 (2006)
6. Gudenkauf, S., Hasselbring, W., Heine, F., Höing, A., Kao, O., Scherp, G.: A Software Architecture for Grid Utilisation in Business Workflows. In: MKWI. GITO-Verlag, Berlin (2008)
7. Gudenkauf, S., Hasselbring, W., Heine, F., Höing, A., Scherp, G., Kao, O.: Bis-Grid: Business Workflows for the Grid. In: CGW 2007 Proceedings, Krakow, Poland, pp. 86–94. ACC CYFRONET AGH (2008)
8. Leymann, F.: Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1201–1217 (2006)
9. Ragouzis, N., Hughes, J., Philpott, R., Maler, E., Madsen, P., Scavo, T.: Security Assertion Markup Language (SAML) V2.0 Technical Overview (February 2007) (Working Draft), http://www.oasis-open.org/committees/download.php/22553/sstc-saml-tech-overview-2%200-draft-13.pdf
10. Slomiski, A.: On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. Concurr. Comput.: Pract. Exper. 18(10), 1229–1241 (2006)